White Paper

SAFE: Worm Mitigation

Introduction

The past two years have seen the release of four major worms on the Internet—CodeRed, NIMDA, SQL Slammer, and MS Blaster. While other worms have been released during this time period that target operating systems such as Solaris and Linux, these four worms have had the greatest impact on the Internet. Each worm targeted specific vulnerabilities in the Microsoft Windows operating system platform, exploited those vulnerabilities to install itself on target systems, and then used these newly infected systems as launching points of attack against other systems. While the underlying exploits used to achieve access to the target hosts varied between these worms, the methods used to mitigate and contain the infection remained the same. This document discusses these containment and mitigation techniques and technologies.

Worm Function

Typically, worms are self-contained programs that attack a system and try to exploit a vulnerability in the target. Upon successful exploitation of the vulnerability, the worm copies its program from the attacking host to the newly exploited system to begin the cycle again. A virus normally requires a vector to carry the virus code from one system to another. The vector can either be a word processing document, an e-mail, or an executable program. The main element that distinguishes a computer worm from a computer virus is the requirement of human interaction to facilitate the spreading of a virus.

Worm Anatomy

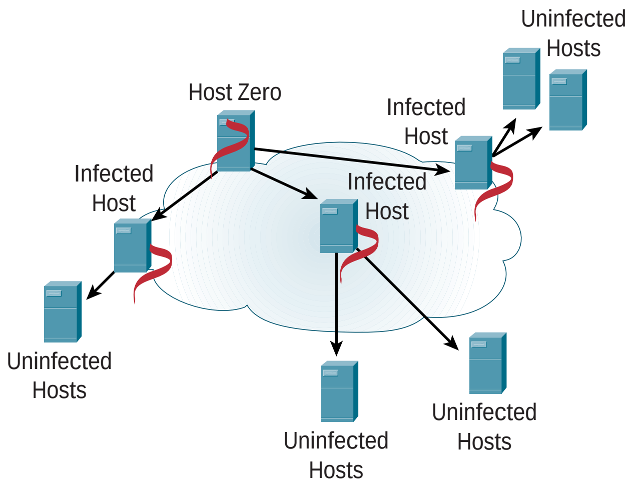
Worms are comprised of three primary components:

- The enabling exploit code
- A propagation mechanism
- A payload

The enabling exploit code is used to exploit a vulnerability on a system. Exploitation of this vulnerability provides access to the system and the ability to execute commands on the target system. Once access has been obtained through the enabling exploit, the propagation mechanism is used to replicate the worm to the new target. The method used to replicate the worm can be achieved through the use of Trivial File Transfer Protocol (TFTP), FTP, or another communication method. Once the worm code has been brought to the new host, the cycle of infection can be started again.

Some worms also contain "payloads," including additional code, to further exploit the host, modify data on the host, or to change a Web page. A payload is not a required component, and in many cases, the worm code itself can be considered the payload. Figure 1 shows a typical worm infection. In this figure, the first host that the worm infects can be thought of as "Host Zero" (similar to "Patient Zero" in biological epidemics). Host Zero seeks out and finds other vulnerable hosts to exploit and infect with the worm code. These infected hosts then seek out other uninfected systems to attack and exploit if possible.

Figure 1
Worm Infection and Propagation



Worm Background and Chronology

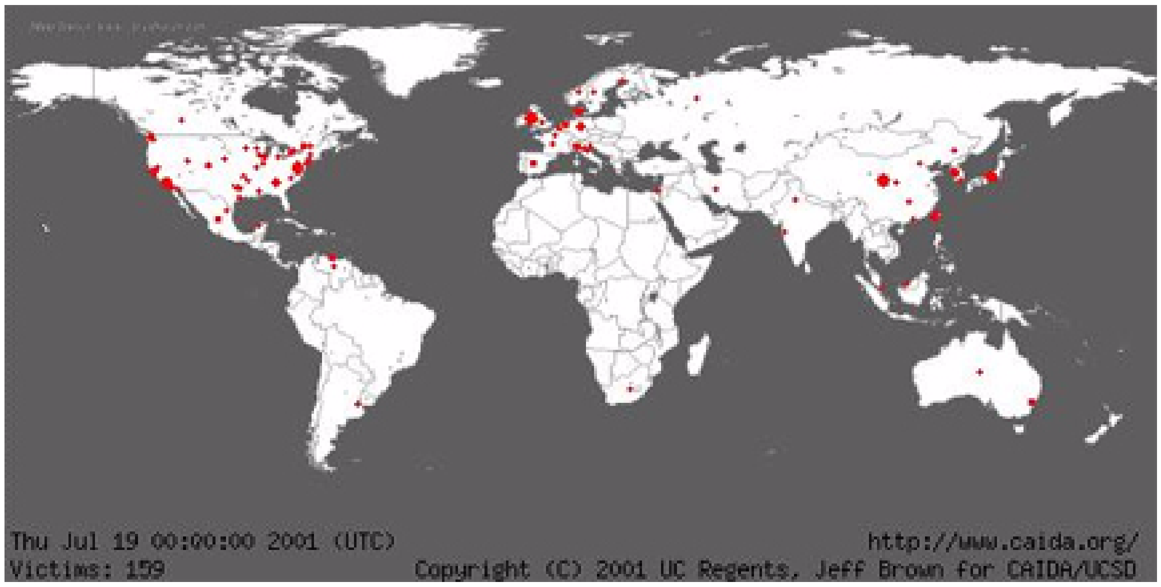
Network worms and viruses have existed for more than 20 years. One of the most famous worm programs to affect the Internet was the Morris worm in November 1988. This worm exploited vulnerabilities in the *finger* and *sendmail* programs of UNIX systems. At that time, the Internet, then called DARPAnet, consisted of approximately 60,000 hosts. This worm infected approximately 10 percent of DARPAnet and caused significant outages and slowdowns of mail servers across the DARPAnet. Since then, many smaller worm infections have appeared across the Internet, and the landscape of the Internet has changed dramatically.

In May 2001, a worm released on the Internet began to infect hosts running Microsoft's Windows Internet Information Server (IIS), as well as systems running the Solaris operating system. This worm infected Solaris systems by exploiting a buffer overflow vulnerability in the Sun Solstice AdminSuite *sadmind* program. With successful exploitation of the vulnerability on the Solaris host, the worm copied the rest of its code to the exploited Solaris system and began searching for other systems to exploit and infect. The worm also contained code to search for hosts running the Windows IIS software and to exploit those hosts using the Microsoft IIS Extended Unicode Directory Traversal vulnerability (http://www.securityfocus.com/bid/1806), which allowed the worm to replace the default home page of the Web server. This worm became known as the *sadmind/IIS* worm. A characteristic of the sadmind/IIS worm that helped slow its spread was that it relied on two disparate operating systems and two completely different vulnerabilities.

Two months later, in July 2001, a new worm infection appeared that would significantly raise awareness of the threat posed by these malicious software programs. CodeRed targeted Microsoft Windows IIS using a vulnerability in the IIS Indexing Service (http://www.securityfocus.com/bid/2880). While the first variant of this worm did little damage due to a flaw in the random number generator code used to generate addresses of hosts to exploit, a second variant appeared with the flaw fixed. This worm, CodeRedv2, spread quickly and became the most widespread and damaging worm to hit the Internet since the Morris worm. CodeRedv2's success as a worm relied on the fact that the worm only exploited the vulnerability in the IIS

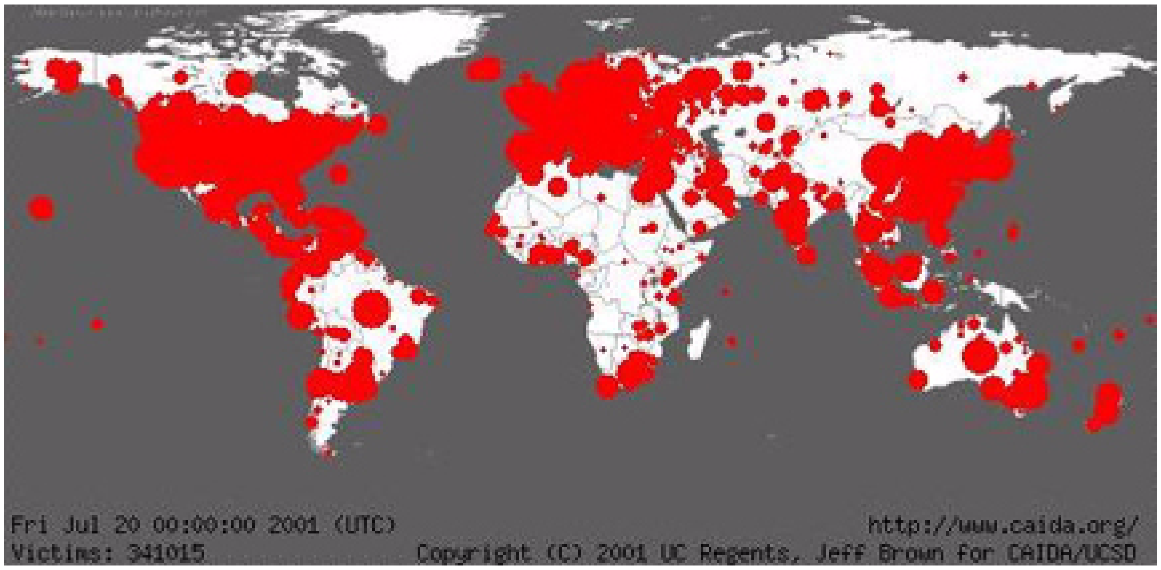
Indexing Service as a means of gaining access to the host. This, coupled with the wide deployment of IIS as well as the large number of unpatched IIS Web servers, contributed to the quick and wide-ranging spread of the worm. An estimated total of 360,000 hosts were infected within a period of 14 hours. Figures 2 and 3 show the spread of CodeRedv2 at the estimated beginning of and end of the first day of infection, respectively.

Figure 2
Initial CodeRedv2 Infection Spread


Thu Jul 19 00:00:00 2001 (UTC)
Victims: 159
http://www.caida.org/
Copyright (C) 2001 UC Regents, Jeff Brown for CAIDA/UCSD

Source: http://www.caida.org/ analysis/security/code-red/coderedv2_analysis.xml

Figure 3
CodeRedv2 Infection Spread after 24 Hours


Fri Jul 20 00:00:00 2001 (UTC)
Victims: 341015
http://www.caida.org/
Copyright (C) 2001 UC Regents, Jeff Brown for CAIDA/UCSD

Source: http://www.caida.org/ analysis/security/code-red/coderedv2_analysis.xml

CodeRedv2 temporarily replaced the home page of the Web servers it struck with a new page. After a few hours, the original home page was restored. Unlike previous worms, CodeRed (both the first variant and CodeRedv2) resided completely in memory and did not store any code on the disk of the Web server. Additionally, the code of the worm indicated that it was programmed to begin a packet-flooding *denial-of-service* (DoS) attack against a hard-coded IP address (at the time, this address was the IP address of the White House Web server www.whitehouse.gov) from the twentieth through the twenty-seventh of each month.

Shortly after the primary infection of CodeRedv2, another variant of the CodeRed worm appeared—CodeRed II. Like its predecessors, CodeRed II infected Microsoft IIS Web servers using the vulnerability in the IIS Indexing Service. However, unlike the CodeRed and CodeRedv2 worms, CodeRed II was not memory-resident. It stored a copy of itself on a server's disk and configured the server to start the worm after a reboot. Additionally, the worm copied the cmd.exe program to the IIS scripts directory, providing a ready-made back door to the server for any attacker to use.

Two months after CodeRed struck the Internet, another large-scale worm struck. Unlike CodeRed, NIMDA was a hybrid worm; rather, it contained the characteristics of both a worm *and* a virus. NIMDA spread using several vectors:

- Through e-mail as an attachment (virus)
- Through network shares (worm)
- Through JavaScript by browsing compromised Web sites (virus)
- Through infected hosts actively scanning for additional exploitable hosts (worm)
- Through infected hosts actively scanning for back doors created by the CodeRed and sadmind/IIS worms (worm)

Unlike CodeRed, NIMDA did not appear to exhibit intentional destructive capabilities. To date, NIMDA's activities have been restricted to its self-propagation, which has the side effect of a DoS attack.

After the NIMDA infection subsided, the Internet saw the appearance of smaller infections of worms. In January 2003, a new worm infected the Internet at such a high rate that it was classified as a “flash worm.” This worm, termed *SQL Slammer,* once again targeted Microsoft Windows servers. However, this worm targeted servers running Microsoft SQL Server software. The vulnerability exploited by SQL Slammer had been published in July 2002, and a patch from Microsoft was available at that time as well. Even though this patch was available for almost six months, SQL Slammer spread with incredibly high efficiency.

SQL slammer was a 376-byte User Datagram Protocol (UDP)-based worm that infected Microsoft SQL servers through UDP port 1434. Because of its small size, the worm was contained in a single packet. The fast scanning rate of SQL Slammer was also achieved because of this small size, as well as the fact that the worm was UDP-based. The worm did not have to complete a handshake (necessary with TCP-based worms) to connect with a target system. SQL Slammer reached its full scanning rate of 55 million scans per second within three minutes of the start of the infection and infected the majority of vulnerable hosts on the Internet within 10 minutes of the start of the infection with an estimated 250,000-300,000 infected hosts overall. A major consequence of such a fast scanning rate was that edge networks were overwhelmed by the amount of traffic generated by the worm. SQL Slammer’s doubling rate was approximately 8.5 seconds (the number of hosts infected with the worm doubled every 8.5 seconds). In contrast, CodeRed II’s doubling rate was on the order of 37 minutes and CodeRed II required approximately 14 hours to reach a saturation impact limit of 360,000 hosts. CodeRed II also created a DoS situation against several networks by inadvertently crashing routers by injecting the exploit code to the router’s Web server.

The next major worm infection after SQL Slammer subsided occurred in August 2003. This worm exploited a flaw in the remote-procedure-call (RPC) code dealing with message exchange over TCP/IP in Microsoft Windows systems. This flaw was a stack-based buffer overflow occurring in a low-level Distributed Component Object Model (DCOM) interface within the RPC process listening on TCP/IP ports 135, 139, and 445. The DCOM protocol enables Microsoft software components to communicate with one another and includes Internet protocols such as HTTP directly over a network. Successful exploitation of this vulnerability allows an attacker to run code with local system privileges, which is equivalent to the UNIX *root* account privileges.

Once MS Blaster successfully exploits a host, it attempts to upload a copy of the worm program to the newly exploited host. MS Blaster uses TFTP to copy the worm program from the attacking host to the target system. MS Blaster also starts up a cmd.exe process and binds it to TCP port 4444 of the newly exploited system. This provides any attacker with direct command line access at the local system privilege level, as discussed above. To access the system, the attacker need only Telnet to TCP port 4444 on the exploited host. If the worm is successful in copying the MS Blaster program to the target, the worm exploit code modifies the system registry to ensure that the worm is restarted if the system reboots. It then launches the worm program on the newly exploited host to begin the cycle again, starting with scanning for more exploitable hosts. MS Blaster also contained code for a DoS attack. This particular attack was targeted at *windowsupdate.com* and would occur based on a particular date—if the month was between January and August, the worm would try and DoS the *windowsupdate.com* address between the sixteenth of the month and the end of the month; if the month was September through December, the worm would send the DoS traffic every day. The DoS traffic consisted of fifty 40-byte HTTP packets to TCP port 80 of the *windowsupdate.com* address. If the worm could not resolve the *windowsupdate.com* address, the destination address of the packets would be set to 255.255.255.255, the local broadcast address.

Worm Mitigation

Worm mitigation requires diligence on the part of system and network administration staff. Coordination between system administration, network engineering, and security operations (SECOPS) personnel is critical in responding effectively to a worm incident.

Typical incident response methodologies can be subdivided into three major categories.

1. Reaction
2. Recovery
3. Post-mortem

Like other incident response situations, the actual response to a worm infection occurs at the "reaction" phase and can be further broken down into four subphases:

1. Containment
2. Inoculation
3. Quarantine
4. Treatment

The following paragraphs provide overviews of these four phases. Information about the network implementation of these phases is provided later in the paper.

Containment

The containment phase involves limiting the spread of a worm infection to areas of the network that are already affected. This requires the compartmentalization and segmentation of the network in order to slow down or, more preferably, stop the worm from further infecting other systems. Networks built on the module principles discussed in the SAFE white papers have readily available "choke points," where intermodule traffic can be easily restricted and used to contain a worm infection. This should be done in a manner that prevents currently infected hosts from targeting and infecting systems outside the network. This requires the use of ingress as well as egress filters on routers and firewalls at various control points within the network.

Inoculation

In parallel with the containment phase, all uninfected systems should be patched with the appropriate vendor patch for the vulnerability. With the worm infection contained, or at the least, significantly slowed down, the inoculation process further deprives the worm of any available targets. A network scanner can help identify potentially vulnerable hosts on the network. The mobile environment prevalent on networks today poses significant challenges to SECOPS and network operations personnel. Laptops are routinely taken out of the "secure" environment of the corporate LAN and connected to potentially "insecure" environments, such as home networks and customer networks. Without proper patching of the system, a laptop can be infected with a worm or virus and can then bring it back into the "secure" environment of the corporate LAN, where it can infect other systems.

Quarantine

The quarantine phase involves tracking down and identifying infected machines within the contained areas and disconnecting, blocking, or removing them. This isolates these systems appropriately for the final phase—treatment.

Treatment

During the treatment phase, actively infected systems are disinfected of the worm. This can involve simply terminating the worm process and removing any modified files or system settings that the worm introduced, and patching for the vulnerability the worm used to exploit the system. In other cases, a complete re-install of the system may be warranted to ensure that the worm and its byproducts are removed.

Worm Mitigation Axioms

This section outlines general best practices that apply specifically to worm mitigation. These practices are outlined here to avoid duplication throughout the case studies discussed below.

Keep All Systems Patched for Security Vulnerabilities

The most effective way to mitigate any worm and its variants is to patch all vulnerable systems. This is difficult with uncontrolled user systems in the local network, and even more troublesome if these systems are remotely connected to the network via a VPN or remote access server (RAS). Administering numerous systems involves the creation of a standard software image that is deployed on new or upgraded systems. These images may not contain the latest patches; continually remaking the image in order to integrate the latest patch may quickly become administratively time-consuming. Pushing patches out to all systems requires that those systems be connected in some way to the network, which may not be possible. One solution to management of critical security patches is to create a central patch server that all systems must communicate with after a set period of time. Any patches that are not applied to a host that is available on the patch server would be automatically downloaded and installed without user intervention. However, determining which devices are exploitable can be simplified by the use of security auditing tools that look for vulnerabilities.

Use Host-Based Intrusion Detection Systems (HIDSs) to Protect End Systems

An HIDS operates by detecting attacks occurring on a host on which it is installed. It works by intercepting OS and application calls, securing the OS and application configurations, validating incoming service requests, and analyzing local log files for after-the-fact suspicious activity. Typically, there are two modes of operation for HIDS—monitor (alarm only) and enforce. HIDS performs many security functions, including:

- Analyzing incoming traffic and, through the use of generic rules and known attack signatures, determining if an attack is present
- Analyzing the server's actions to determine if they reflect its normal mode of operations
- General OS protection, including buffer overflow prevention and binary modification

HIDSs can have the same problem with exploitation mitigation as discussed previously for applying system patches. However, HIDS clients can be significantly easier and less obtrusive to install on running systems. Additionally, HIDSs are less likely to require system interruptions or reboots. To target specific systems for HIDS installation for the current problem, a network security scanner can be used to identify specific systems that would be considered ideal candidates for installation of the HIDS software.

Deploy Network-Based Intrusion Detection Systems (NIDSs) to Detect Worm Activity

An NIDS operates by first detecting an attack occurring at the network level and then either taking a corrective action itself or notifying a management system where an administrator can take action. Attacks are discovered by looking for their signatures in traffic flows in the network. Attack detection triggers NIDSs to send an alarm and then take a preconfigured action. The two possible actions are “shunning” and TCP resets. Since an NIDS is not in the data path, meaning it receives a copy of a packet as it traverses through the network verses routing the packet, it cannot filter the first packet in an attack. Subsequent packets can be filtered via shunning, a feature that modifies the upstream access-control device to block any further access from the IP address of the attacking system. TCP resets attempt to tear down the TCP connection by sending a fabricated reset that appears to be from the receiving device to the attacking device.

Anomaly IDSs also provide for the detection of possible infections by using statistical analysis of network traffic to identify worms. An anomaly IDS can provide quicker identification and detection of worm activity than signature-based NIDSs. This is due in part to the capability of anomaly-based IDSs to detect small deviations from normal activity through statistical analysis of network traffic.

Refer to the *SAFE: IDS and Logging In-Depth* white paper for more information about implementing shunning on a network—there are special considerations to consider when using this feature.

Use Access Control to Restrict Worm Traffic

Stateful firewalling provides many security features that proactively mitigate worms. First, the stateful inspection engine can control connection attempts at a granular level by validating proper protocol adherence. This filtering could be used to allow only inbound connections to a server, and at the same time disallow that server to initiate outbound connections, limiting a worm’s ability to self-propagate. This is particularly applicable for DMZ server deployments. As discussed in SAFE, servers normally do not need the ability to establish outbound connections—in most cases, they only need to respond to incoming requests. Second, stateful firewalling can limit the number of permitted inbound connections to a server so that the server will not become overwhelmed.

Ingress filtering is typically carried out by access control on the perimeter of the network. It is used to block access to hosts and services that should not be publicly available. For instance, it is a security best practice to disallow incoming connection requests to hosts or networking devices, unless those hosts or devices are actively participating in providing a publicly accessible service.

Egress filtering is also typically carried out by access control on the perimeter of the network. This filtering blocks a local host’s access outbound out of the network. Devices that do not need outbound Internet access, such as the majority of the networking devices in the network or servers that only serve the internal environment, should not be allowed to initiate outbound connections. Additional layers of egress filtering in the network (besides at the WAN edge) could also be used to disallow an infected public server (or its entire segment, in the case of a server farm) from infecting private internal servers that were protected by the edge ingress filtering. For more information on access control and filtering, please refer to the SAFE white papers.

Use Private VLANs to Protect Critical End Systems Within a VLAN

Private VLANs work by limiting the ports within a VLAN that can communicate with other ports in the same VLAN. Typically, private VLANs are deployed so that the hosts on a given segment can only communicate with their default gateways and not the other hosts on the network. For instance, if a server is compromised by a worm, it will not be able to initiate infection attempts to other servers in the same VLAN, even though they exist in the same network segment. This access control is carried out by assigning hosts to either an isolated port or a community port, and is an effective way to mitigate the effects of a single compromised host. Isolated ports can only communicate with promiscuous ports (typically the router). Community ports can communicate with the promiscuous port and other ports in the same community. For more information on private VLANs, visit:

http://www.cisco.com/warp/public/473/90.shtml

Network-Based Application Recognition (NBAR)

NBAR is a classification engine in Cisco IOS® Software that can recognize several application level protocols, including HTTP via URL/MIME type and protocols that use dynamic port assignments. Once NBAR has classified the traffic, appropriate quality of service (QoS) policies can be applied to the traffic classes to handle the traffic appropriately. Mission-critical applications can be given preferential treatment in the allocation of bandwidth, while non-mission-critical traffic can be marked for best-effort service, policed or blocked. This technology can be useful in slowing down or preventing worm traffic by using NBAR to police traffic to specific ports and, depending on the policy, either limiting the bandwidth associated with this traffic or preventing it altogether.

Identify Worm Infected Systems

Use NetFlow to Track Worm Attack Hosts

NetFlow provides network engineers and administrators with several applications that track and identify IP traffic. These applications include traffic accounting, network monitoring, DoS monitoring, and data mining. This information is particularly useful in tracking down worm attack hosts, as well as monitoring the progression of a worm through a network. By using NetFlow to identify worm attack hosts, network operators can implement access-control lists (ACLs) to block traffic originating from the attacking systems.

Use Sink-Hole Routers to Identify Infected Systems

Sink-hole routers are typically used by a service provider to redirect malicious IP traffic to a single IP address where the traffic can be examined in greater detail. Service providers can use this concept to identify networks and individual hosts where worm traffic is originating. This concept can also be applied within an enterprise architecture environment to identify hosts that are infected by a worm and are actively seeking additional target systems. Setting up a sink-hole router will assist in determining which systems in the environment are infected when NIDS is not available, either due to insufficient resources to deploy NIDS or other architectural constraints. This works by using addresses not yet allocated by the Internet Assigned Numbers Authority (IANA) that some worms will inadvertently attempt to exploit. The sink-hole router advertises these networks locally (only), and any attempts at reaching them will then be routed to the router. Once received, they can be logged and discarded. The logs will provide a list of infected hosts. For more information on how to configure this, visit:

http://www.cisco.com/public/cons/isp/security/

Use Committed Access Rate (CAR) to Rate-Limit Traffic

CAR can be used to rate-limit traffic based on a set of criteria, and provides for configurable actions such as transmit, drop, set precedence, or set QoS group when the traffic meets or exceeds the rate limit. These criteria include such metrics as incoming interface, IP precedence, QoS group, and IP access list criteria. CAR is typically applied to traffic on a router and performs two basic QoS functions:

- Bandwidth management through rate-limiting
- Packet classification

By using CAR, network engineers can classify and control traffic into and out of their networks, thereby providing a capability to prevent the bandwidth saturation seen by several service providers during the SQL Slammer worm propagation. For more information on CAR, visit:

http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a00800c 75ce.html

Implement Virus Scanning Throughout the Network

Virus scanning software provides real-time host attack mitigation against known malicious code and viruses. As with NIMDA, viruses may have multiple paths into the system, including e-mail, browsing, and file exchange. These paths may themselves intrinsically provide multiple vectors into the system. Web pages, for instance, may use ActiveX, Java, and JavaScript to load remotely available code in order to provide additional capabilities. However, all of these mechanisms are entry points for executing malicious code on the system. In most cases, the user is prompted to allow the remote code to execute. Due to lack of user education, most will click "yes" without hesitation. Worse yet, older Web browsers will not even prompt the user and will execute the code automatically. In order for virus scanning to be successful, the following should be completed at regular intervals:

- Routine host local file scanning
- Routine virus list and signature updating
- Routine monitoring of alerts generated by the host scanners

Virus scanning should be conducted both at the inbound and outbound mail gateways, as well as on the end hosts where e-mail is read. Virus scanning should be conducted as a complement to HIDS deployment. While HIDS may not be deployable across all systems on the network (due to limitations in resources or limitations in the HIDS), antivirus scanning on mail gateways can be accomplished with relative ease and provides significant protections against e-mail-borne attack code.

Implement Denial of Service Best Practices

CodeRed and MS Blaster each contained code that directed the worms to execute a distributed denial of service attack against www.whitehouse.gov and windowsupdate.com respectively. Additionally, the spread of a worm can be fast enough that the traffic generated by the worm itself is enough to create a denial of service condition as was demonstrated by SQL Slammer. As worms represent an effective way to spread denial of service attack agents across a wide range of networks in a short period of time it is critical that, along with the best practices described above network personnel should also implement the necessary steps to help mitigate these types of attacks. These steps include securing the network infrastructure devices such as turning off unnecessary services, restricting access to management ports. Additionally, the use of QoS as well as other traffic policing methods to help restrict the spread of the infecting code should be implemented.

If a worm infection does occur on the network and it is determined that the worm code contains a denial of service attack against a website, DNS poisoning may mitigate the effects of that attack. DNS poisoning includes adding an address record in the local DNS hosts for the target system such that the IP address resolves to a non-existent host or the localhost address. The intention is to redirect the worm traffic and prevent it from reaching the actual target system. Using the localhost address results in the worm executing the DoS attack against the worm host itself. Care must be used when implementing DNS poisoning. Tests with MS Blaster utilizing DNS poisoning resulted in unexpected behavior from the worm.

Case Studies

The following case studies cover the four major worms that have infected the Internet since 2001. In each case, the mitigation techniques are most applicable to containment and quarantine. Not all axioms are applied across each case study, only the axioms that were deemed most effective. However, just because a mitigation method is not discussed in a case study does not indicate that the mitigation method is inappropriate in that particular situation.

CodeRed v2

The following links provide information regarding infection mitigation of the CodeRed worm and its variants on Microsoft and Cisco Systems products:

- http://www.cisco.com/warp/public/707/cisco-code-red-worm-pub.shtml
- http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-033.asp

Many vendor products install and use IIS to provide Web access for remote management and reporting, and these will also be vulnerable unless patched. If it is not possible to patch all systems in a timely manner, consider deploying the technologies discussed in the “Worm Mitigation Axioms” section above. Consider using these technologies proactively to mitigate future attacks by variants of CodeRed or other attacks altogether.

Host-Based Intrusion Prevention Systems (IPS)

In the case of CodeRed and its variants, an HIPS secures IIS by disabling the indexing service. It will also send an alarm to the centralized monitor console indicating that an exploitation attempt was intercepted.

NIDS

Shunning is not recommended for attack mitigation of CodeRed. Since CodeRedv1 and v2 contain the attack in a single packet, NIDS cannot stop the attack. NIDS does, however, provide visibility by alarming when CodeRedv1 and v2 attacks traverse the network. NIDS is able to stop CodeRed II attacks with high probability through the use of TCP resets, as CodeRed II uses multiple packets. For more information on NIDS, visit:

http://www.cisco.com/go/ids

Access Control

As it pertains to CodeRed, incoming HTTP connections would be blocked from accessing any possibly exploitable user systems or non-publicly available Web servers. These same filters, however, had to allow access to a publicly available Web presence or e-commerce server. Ideally, these public servers were under tight administrative control and had the latest patches, but in too many cases, they were not. Ingress filtering would, in effect, block CodeRed exploitation attempts targeted at user systems.

If a device is compromised, egress filtering prevents the system from successfully executing a distributed DoS (DDoS) attack against an external network. It also may help prevent the compromised system from infecting other hosts outside the network perimeter, since the traffic will be intercepted and dropped at the perimeter of the network. This guards against the DDoS attack flooding the Internet link and interfering with legitimate inbound or outbound traffic.

NBAR

NBAR recognizes the CodeRedv1 and v2 URL request, but not the CodeRed II URL request. This is because CodeRed II spreads the GET request over multiple packets, and NBAR typically only inspects the first packet. Unlike an NIDS, NBAR can immediately classify CodeRedv1 and v2 traffic and drop the packet before reaching the server, and can be used inbound and outbound to mitigate CodeRed's effects. For more information on NBAR, visit:

http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121newft/121limit/121e/121e2/nbar2e.htm

NIMDA

NIMDA infected vulnerable target systems through several means, including e-mail, Web browsing, and actively looking for and exploiting the back doors left behind by the CodeRed II worm. The most effective mitigation technique against NIMDA is patching all vulnerable systems, which is difficult with uncontrolled user systems in the local network and even more troublesome if the systems are remotely connected to the network via a VPN or RAS. However, determining which devices are exploitable can be simplified by the use of security auditing tools that look for vulnerabilities in server systems. For local workstations, the PC's browser and e-mail client may need to be patched. The following link provides information regarding infection mitigation on Microsoft products:

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-044.asp

Antivirus

Additional NIMDA mitigation methods included updating all virus scanning software with the latest virus lists, running local scans on any systems where infection is suspected, and determining which devices in the network were still infected or vulnerable, in case they were missed during the patching and virus scanning. These tasks can be carried out with network scanners, as well as with analyzing alarms received from an IDS.

HIDS

When a worm attempts to compromise an HIDS-protected Web server, the attack will fail and the server will not be compromised. HIDS blocks the wormlike infection methods (spreading via Microsoft IIS vulnerabilities) by locking down the Web server. HIDS also prevents directory traversal and remote code execution attacks as well as unauthorized changes to Web content, thus limiting the capability of the worm to alter Web pages in order to spread itself to other servers. Finally, HIDS will prevent the Web server from being compromised via HTTP and IIS exploits through attack signature detection. The following HIDS rules prevent NIMDA from succeeding:

- IIS Directory Traversal
- IIS Directory Traversal and Code Execution
- IIS Double Hex Encoding Directory Traversal

Note that the viruslike, manual infection methods, such as opening an e-mail attachment, manually executing an infected file, and browsing to an infected Website, are not blocked by HIDS. These can be mitigated by the security best practices covered in SAFE, including virus scanning and thorough education of the user base. For instance, administrators should not run client e-mail applications or browse the Web on production Web servers. It is also a best practice not to run network shares on pubic servers.

NIDS

An NIDS identifies many of the Web application attacks used by the NIMDA worm and provides details about the affected and compromised hosts.

The following Cisco IDS Network Sensor alarms will be triggered by the NIMDA worm:

- WWW WinNT cmd.exe Access (SigID 5081)
- IIS CGI Double Decode (SigID 5124)
- WWW IIS Unicode Attack (SigID 5114)
- IIS Dot Dot Execute Attack (SigID 3215)
- IIS Dot Dot Crash Attack (SigID 3216)

NIDS operators will not see an alarm that identifies NIMDA by name. They will see a series of these alarms as NIMDA tries different exploits to compromise the target. These alarms will identify the source address of hosts that have been compromised and should be isolated from the network, cleaned, and patched.

SQL Slammer

The first and most effective manner in which to mitigate the SQL Slammer worm is to patch all vulnerable systems. As with CodeRed and NIMDA, patching may be difficult with uncontrolled user systems in the local network and even more troublesome if the systems are remotely connected to the network via a VPN or RAS. However, determining which systems are exploitable can be simplified by using security-auditing tools that look for vulnerabilities, such as the ones listed at the end of this paper.

Scanning for systems that may be running the Microsoft SQL Resolution Service provides for quick identification of potentially vulnerable hosts. After these vulnerable systems are identified, they can be patched to remove the vulnerability. Monitoring log files for hits on the ACLs discussed above can identify hosts already infected by the SQL Slammer worm. The following links provide information about infection mitigation on Microsoft SQL products:

- http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp
- http://www.microsoft.com/technet/security/virus/alerts/slammer.asp

Access Control

Stateful firewalling provides numerous security features to proactively mitigate the SQL Slammer worm. This filtering can be used to allow only inbound connections to a SQL server, and at the same time, to disallow that SQL server from initiating outbound connections—limiting the ability of the worm to self-propagate.

As discussed in the SAFE Blueprint, SQL servers do not normally need the ability to establish outbound connections to external systems. In most cases, they need to respond only to incoming SQL requests. In addition, stateful firewalling has the capability of limiting the number of permitted inbound connections to a server so that the server does not become overwhelmed. In the case of SQL Slammer, this limiting blocks inbound exploitation connection attempts.

If stateful firewalling is not possible, the next most effective method to contain this worm is the application of ingress and egress filters or ACLs blocking port 1434 UDP. Ingress filtering is typically performed by access control on the perimeter of the network. Incoming SQL connections would be blocked from accessing any possibly exploitable user systems or non-publicly available SQL servers. These same filters, however, would need to allow access to a publicly available SQL presence or e-commerce server. Ideally, the public servers are under tight administrative control and have the latest patches. Ingress filtering would, in effect, block SQL Slammer exploitation attempts targeted at user systems.

Egress filtering prevents compromised SQL servers from launching a DDoS attack against an external network because the traffic is intercepted and dropped at the perimeter of the network. This setup also guards against the DDoS attack flooding the Internet link and interfering with legitimate inbound or outbound traffic. Additional layers of egress filtering in the network, in addition to those at the WAN edge, can also be used to disallow an infected public SQL server (or its entire segment for the case of an SQL farm) from infecting private internal servers that are protected by the edge ingress filtering. For more information about access control and filtering, see the SAFE Blueprint white papers. A sample ingress and egress filter rule to add to existing ACLs is provided below.

Cisco IOS ACLs

Care must be taken when considering whether to use the **log-input** argument to the **access-list** command. It is possible to substantially increase the CPU usage on the router because of the logging on the ACL. If router performance degrades due to the introduction of these ACLs, discontinue the logging on the first ACL.

```
access-list 101 deny udp any any eq 1434 log-input
access-list 101 permit ip any any
```

A more fine-tuned approach would be to create an ACL for the offending SQL Slammer worm traffic and then use class-based policing to drop the packets at the ingress interface.

1. Create ACL

   ```
   access-list 101 permit udp any any eq 1434
   ```

2. Match on ACL and packet length

   ```
   class-map match-all slammer_worm
   match access-group 101
   match packet length min 404 max 404
   ```

3. Use class-based policing to drop matching packets at the ingress interface

   ```
   policy-map drop-slammer-worm
   class slammer_worm
   police 1000000 31250 31250 conform-action drop exceed-action drop violate-action drop
   ```

After further worm infection has been prevented through the use of ingress and egress filters, the next step would be identifying and tracking vulnerable hosts, as well as systems that may already be infected.

HIDS

The HIDS response to SQL Slammer is based on the system it is installed on:

- On the HIDS Management Console (running MSDE), the HIDS Default Manager policy prevents incoming connections to the MSDE on the SQL port.
- On a Microsoft SQL server 2000 system, the HIDS default policy accepts incoming connections to SQL; however, the HIDS buffer overrun logic detects and terminates the Slammer worm's attempt to invade the system.

- On a desktop system, the HIDS Desktop policy prevents incoming connections to the MSDE on the SQL port.

In the case of the SQL Slammer worm the Cisco Security Agent prevented the initial buffer overrun that it exploited, and the default policies provided additional protection at both the server and desktop level against the propagation of this worm.

NIDS

Because the SQL Slammer attack is contained within a single packet, NIDS cannot stop the attack. NIDS does, however, provide visibility by sending an alarm when SQL Slammer attacks traverse the network. For more information about NIDS, visit:

http://www.cisco.com/go/ids

NIDS Attack Signatures

The signatures provided below were added to NIDS systems (Cisco Secure IDS 4215 Sensor, Cisco Secure IDS 4235, and Cisco Secure IDS 4250 Sensor IDS Module) in many modules of the SAFE Blueprint.

SQL Slammer Worm

String: ”\x04\x01\x01\x01\x01\x01.*[.][Dd][Ll][Ll] "

Occurrences: 1

Port: 1434

Recommended alarm severity level:

- High (VPN and Security Management [VMS])
- 5 (UNIX Director)

The latest NIDS signature database is available from Cisco.com.

Private VLANs

Private VLANs restrict a compromised SQL server from initiating infection attempts against other SQL servers in the same VLAN, even though they exist in the same network segment.

NBAR

NBAR can recognize the SQL Slammer worm and can immediately classify the traffic and drop the packet before it reaches the server. It can be used inbound and outbound to mitigate the effects of the SQL Slammer worm. NBAR provides for the creation of a custom protocol to monitor traffic not normally associated with NBAR. The following is an example configuration:

Custom Protocol in NBAR

1. Create custom protocol

   ```
   ip nbar port-map custom-01 udp 1434
   ```

2. Create class-map

   ```
   class-map match-all slammer_worm
   match protocol custom-01
   match packet length min 404 max 404
   ```

3. Use class-based policing to drop the matching packets at the ingress interface

```
policy-map drop-slammer-worm
class slammer_worm
police 1000000 31250 31250 conform-action drop exceed-action drop violate-action drop
```

Sink-Hole Routers

Sink-hole routers assist in determining which systems in the environment are infected with SQL Slammer. The sink-hole router advertises network addresses not yet allocated by IANA so that any attempts to reach those network addresses are routed to the sink-hole router. When these connection attempts are received, they can be logged and discarded. The logs will provide a list of infected hosts.

Unicast Reverse Path Forwarding

The Unicast Reverse Path Forwarding (RPF) feature helps mitigate problems caused by the introduction of spoofed IP source addresses into a network. It works by discarding IP packets that lack a verifiable IP source address. There are two Unicast RPF checking modes:

- Strict checking mode, which verifies that the source IP address exists and is reachable through the input interface
- Exist-only checking mode, which only verifies that the source IP address exists in the Forwarding Information Base (FIB) table

Some customers reported that the SQL Slammer worm used IP packets with spoofed addresses and that the use of the command **ip unicast reverse path forwarding** provided some appreciable measure of relief from the SQL Slammer worm.

NetFlow Configuration

To configure NetFlow on a NetFlow-capable router to track SQL Slammer infected hosts:

```
Router# config t
Router# (config) interface serial 0/1
Router#(config-if) ip route-cache flow
Router#(config-if) exit
Router#(config) exit
Router#
```

After NetFlow has been enabled on the router, the information can be exported to several network management applications. To export NetFlow statistics:

```
Router# (config) ip flow-export 192.168.155.1 700
```

To view NetFlow statistics for port 1434:

```
Router# show ip cache flow | include 059A
```

For more information about how to configure NetFlow, visit:

http://www.cisco.com/en/US/products/sw/iosswrel/ps1826/products_configuration_guide_chapter09186a00800880 f9.html

CAR

By using CAR, network engineers can classify and control traffic into and out of their networks, thereby providing a capability to prevent the bandwidth saturation seen by several service providers during the SQL Slammer worm propagation.

CAR Configuration

Like NetFlow, CAR can be configured on several Cisco routers. The following configuration is an example:

```
Router# (config) access-list 150 deny udp any any eq 1434
Router# (config) access-list 150 permit ip any any
Router# (config) interface fastEthernet 0/0
Router# (config-if) rate-limit input access-group rate-limit 150 8000 1500 20000
conform-action drop exceed-action drop
Router# (config-if) exit
Router# (config) exit
Router#
```

MS Blaster

The most effective method to mitigate the RPC DCOM attack was to patch all vulnerable systems. As with the other worms discussed in this paper, this patching is difficult with uncontrolled user systems in the local network and even more troublesome if they are remotely connected to the network through a VPN or RAS. Identifying vulnerable systems can be simplified through the use of security auditing tools that look for the vulnerabilities that MS Blaster exploits. After vulnerable systems are identified, they can be patched to remove the vulnerability. Information about attack mitigation on Microsoft products is available at:

http://microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp

Access Control

The most effective method to prevent the exploitation of this vulnerability from the Internet is the use of ingress and egress filters, or ACLs blocking access to ports 135 and 139 (TCP and UDP) as well as port 445 (TCP and UDP). Network administration best practices provide no need for these ports to be directly Internet-accessible.

Pertaining to RPC DCOM, incoming connections would be blocked from accessing any possibly exploitable user systems or non-publicly available servers. Ideally, any public servers are under tight administrative control. Also, they have the latest patches, and access to ports 135 and 139 (TCP and UDP) or port 445 (TCP and UDP) would require the use of a VPN tunnel. Ingress filtering would, in effect, block any exploitation attempts from the Internet of the RPC DCOM vulnerability targeted at user systems. However, trusted systems already infected by the RPC DCOM worm and connecting to the corporate network through a VPN could still infect other systems.

Egress filtering prevents a compromised device from launching a reverse Telnet back to the attacker’s system, because the traffic will be intercepted and dropped at the network perimeter. For more information about access control and filtering, see the SAFE Blueprint white papers.

Stateful firewalling provides numerous security features to proactively mitigate the RPC DCOM. First, the stateful inspection engine can control connection attempts at a more detailed level by validating proper protocol adherence. If ingress filtering is not used to block external inbound access to vulnerable systems, then outbound filtering should be used to restrict vulnerable hosts from initiating outbound connections. This limits the ability of an attacker to gain command-line access to the host, and limits the W32/Blaster worm’s capability to spread.

A sample ingress and egress filter rule to add to existing ACLs is provided below.

Cisco IOS Software ACLs

The Cisco IOS Software ACLs for mitigating the RPC DCOM exploits are provided below.

```
access-list 101 deny udp any any eq 135
access-list 101 deny tcp any any eq 135
access-list 101 deny udp any any eq 137
access-list 101 deny tcp any any eq 137
access-list 101 deny udp any any eq 139
access-list 101 deny tcp any any eq 139
access-list 101 deny udp any any eq 445
access-list 101 deny tcp any any eq 445
access-list 101 deny tcp any any eq 593
access-list 101 permit ip any any
```

To block outbound TFTP traffic to prevent the worm version of the exploit from downloading code to a newly infected host:

```
access-list 102 deny udp any any eq 69
```

To block outbound traffic to port 4444/TCP that the exploit uses to provide command line access to a Windows target host:

```
access-list 102 deny tcp any any eq 4444
```

A more fine-tuned approach would be to create an ACL for the offending RPC DCOM traffic, and then use a class-based policing to drop the packets at the ingress interface.

1. Create an ACL

   ```
   access-list 101 deny udp any any eq 135
   access-list 101 deny tcp any any eq 135
   access-list 101 deny udp any any eq 137
   access-list 101 deny tcp any any eq 137
   access-list 101 deny udp any any eq 139
   access-list 101 deny tcp any any eq 139
   access-list 101 deny udp any any eq 445
   access-list 101 deny tcp any any eq 445
   access-list 101 deny tcp any any eq 593
   access-list 101 permit ip any any
   ```

2. Match on ACL and packet length

   ```
   class-map match-all rpc_dcom
   match access-group 101
   ```

3. Use class-based policing to drop matching packets at the ingress interface

   ```
   policy-map drop-rpc-dcom
   class rpc_dcom
   police 8000 1000 1000 conform-action drop exceed-action drop violate-action drop
   ```

HIDS

The Cisco Security Agent response to MS Blaster is based on the system it is installed on:

- The default Cisco Security Agent 4.0 server and desktop policies stop successful execution of this attack
- On servers, the default server policy prevents the SVCHOST from attempting to execute CMD.exe. This prevents the exploit shell code from running.

- On desktop systems, the default desktop policy prevents the SVCHOST from accepting a connection on port 4444. Additional protection is provided by the default policy's prevention of any application from executing CMD.exe.

Because of the sensitive nature of the SVCHOST process in the proper operation of Windows, the Cisco Security Agent detects the overflow but does not terminate the SVCHOST process. Instead, Cisco Security Agent prevents the host from being exploited by terminating the CMD.exe process that the buffer overflow in the SVCHOST process creates because of the exploit.

NIDS

The Cisco IDS Network Security Database (NSDB) includes a signature for the Microsoft Windows RPC DCOM exploit (sig 3327). It is available in IDS signature update S49.

The following custom signature string can be added to address this worm:

Engine	: STRING.UDP
SigName	: MS Blast Worm TFTP Request
ServicePorts	: 69
RegexString	: \x00\x01[Mm][Ss][Bb][Ll][Aa][Ss][Tt][.][Ee][Xx][Ee]\x00
Direction	: ToService

To reduce the number of false positives on this signature, consider restricting this signature's inspection of ports to 137, 139, and 445 only. For registered customers, the following service pack includes this signature for the RPC DCOM exploit.

ftp://ftp-sj.cisco.com/cisco/crypto/3DES/ciscosecure/ids/4.x/IDS-sig-4.1-1-S49.rpm.pkg

Private VLANs

Private VLANs would restrict a compromised Microsoft Server from initiating infection attempts against other Microsoft servers in the same VLAN, even though they exist in the same network segment.

NBAR

NBAR can recognize the Microsoft NetBIOS protocol and protocols that use dynamic port assignments. After the traffic has been classified by NBAR, appropriate QoS policies can be applied to the traffic classes. Unlike NIDS, NBAR can immediately classify the NetBIOS traffic and drop the packet before it reaches the server. NBAR can be used inbound to mitigate the effects of the RPC DCOM exploit.

1. Create class-map

   ```
   class-map match-all msblaster-nbar
   match protocol netbios
   match packet length min 404 max 404
   ```

2. Use class-based policing to drop the matching packets at the ingress interface

   ```
   policy-map drop-msblaster-worm
   class msblaster_worm
   ```

```
   police 1000000 31250 31250 conform-action drop exceed-action drop violate-action drop
```

NetFlow

To view NetFlow statistics for port 135, 139, and 445:

```
Router# show ip cache flow | include 0087
Router# show ip cache flow | include 0089
Router# show ip cache flow | include 01BD
```

For more information about how to configure NetFlow, visit:

http://www.cisco.com/en/US/products/sw/iosswrel/ps1826/products_configuration_guide_chapter09186a00800880 f9.html

CAR

Analysis of the W32/Blaster worm indicates that it contains code to launch a DoS attack against the Website windowsupdate.microsoft.com. CAR can be used to reduce the effects of the attack. Like NetFlow, CAR can be configured on several Cisco routers. The following configuration is an example:

```
Router# (config) access-list 150 permit udp any any eq 135
Router# (config) access-list 150 permit udp any any eq 139
Router# (config) access-list 150 permit tcp any any eq 135
Router# (config) access-list 150 permit tcp any any eq 135
Router# (config) access-list 150 permit udp any any eq 445
Router# (config) access-list 150 permit tcp any any eq 445
Router# (config) access-list 150 deny ip any any
Router# (config) interface fastEthernet 0/0
Router# (config-if) rate-limit input access-group rate-limit 150 8000 1500 20000
conform-action drop exceed-action drop
Router# (config-if) exit
Router# (config) exit
Router#
```

For more information about CAR, visit:

http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a00800c75 ce.html

Conclusion

The technologies discussed in this document not only mitigate the potential damage that known worms can cause, but can also provide a defense against future worms. It is important to remember that security has its place throughout the infrastructure, and the discussed technologies prove this. Protecting a network and its resources against worms like CodeRed, NIMDA, SQL Slammer, and MS Blaster is only the first step. It is necessary to be proactive when it comes to security to protect a network not only against these worms but also against future network attacks.

Establishing a security policy, implementing some of the discussed features, and regular in-house or outsourced posture assessments will secure a network and help to keep it secure. This document has addressed a small sampling of the documented security and network design best practices available from Cisco. For additional information about securing your network, see the SAFE Blueprint at:

www.cisco.com/go/safe

As with any feature, you should ensure that all devices have sufficient CPU resources available before enabling any of the features discussed in this document. Also realize, however, that the increased load brought on by enabling these features is probably significantly less than the load brought on by an internal worm infection.

As a special note, the SAFE Blueprint was released in October 2000. No design or implementation modifications were required to address these four worms. Only NIDS signature updates at regular intervals were necessary to detect the specific exploits associated with each worm. This and other high-profile network exploits constantly provide reminders that designing network security reactively is not recommended. Only by taking a comprehensive approach to network security founded on good security policy decisions can an organization be assured that the risks taken are known, and that virtually any potential threat can be effectively contained.

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

European Headquarters
Cisco Systems International BV
Haarlerbergpark
Haarlerbergweg 13-19
1101 CH Amsterdam
The Netherlands
www-europe.cisco.com
Tel: 31 0 20 357 1000
Fax: 31 0 20 357 1100

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
www.cisco.com
Tel: 408 526-7660
Fax: 408 527-0883

Asia Pacific Headquarters
Cisco Systems, Inc.
Capital Tower
168 Robinson Road
#22-01 to #29-01
Singapore 068912
www.cisco.com
Tel: +65 6317 7777
Fax: +65 6317 7799

Cisco Systems has more than 200 offices in the following countries and regions. Addresses, phone numbers, and fax numbers are listed on the
Cisco Web site at www.cisco.com/go/offices

Argentina • Australia • Austria • Belgium • Brazil • Bulgaria • Canada • Chile • China PRC • Colombia • Costa Rica • Croatia Czech Republic • Denmark • Dubai, UAE • Finland • France • Germany • Greece • Hong Kong SAR • Hungary • India • Indonesia • Ireland Israel • Italy • Japan • Korea • Luxembourg • Malaysia • Mexico • The Netherlands • New Zealand • Norway • Peru • Philippines • Poland Portugal • Puerto Rico • Romania • Russia • Saudi Arabia • Scotland • Singapore • Slovakia • Slovenia • South Africa • Spain • Sweden Switzerland • Taiwan • Thailand • Turkey • Ukraine • United Kingdom • United States • Venezuela • Vietnam • Zimbabwe

ETMG 203149—RD 12.03